DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Primer Trabajo Práctico

14 de Abril de 2010 — Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Bianchi, Mariano	92/08	bianchi-mariano@hotmail.com
Brusco, Pablo	527/08	pablo.brusco@gmail.com
Di Pietro, Carlos Augusto Lyon	126/08	cdipietro@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Ejercicio 1

1.1. Introducción

El primer problema del presente trabajo consistió en la implementación de un algoritmo capaz de dar solución a la ecuación

$$b^n \ mod \ (n) \tag{1}$$

haciendo uso de alguna de las técnicas algorítmicas aprendidas hasta el momento en la materia. Asímismo, la consigna dictaba que la complejidad final del algoritmo debería ser menor a $O(n)$. En pos de cumplimentar lo pedido se decidió usar la técnica de *Dividir & Conquistar*[1] para desarrollar el algoritmo. Esta técnica se caracteriza principalmente en dividir la instancia de un problema en instancias más pequeñas, atacar cada una de ellas por separado y resolverlas, para finalmente juntar sus resultados y así producir el resultado final.

1.2. Explicación

La primera solución que se piensa casi de manera intuitiva es la de mutliplicar n veces el número b y luego hallar el resto de dividir ese resultado por n.

$$\underbrace{b.b.b \ldots b.b.b}_{n \ veces} \ mod \ (n) = b^n \ mod \ (n) \tag{2}$$

Sin embargo, la complejidad ese algoritmo es $O(n)$, ya que se realizan n multipicaciones y 1 división, razón por lo cual no cumple con lo pedido en la consigna. Además, puesto que ese algoritmo realizar sucesivas multipliciones de b, cabe la posibilidad de que el valor en que se va acumulando el resultado sobrepase el máximo número entero representable en la computadora produciedo así un overflow y obteniéndose en consecuencia un resultado incorrecto.

Analizando la falla del algoritmo anterior, se observa que lo que se debe evitar es calcular directamente el resultado de b^n ya que ese número podría ser excesivamente grande.
Con esto en mente veamos una manera más conveniente de abordar el problema. Es claro, por definición de congruencia, que resover la ecuación en (1) equivale a hallar el x tal que:

$$b^n \ \equiv \ x \ (n) \ , \quad con \ 0 \ \leq \ x \ < \ n \tag{3}$$

Luego, asumamos por un momento que podemos tener gratis un k tal que:

$$b^{n/2} \ \equiv \ k \ (n) \ , \quad con \ 0 \ \leq \ k \ < \ n \tag{4}$$

Entonces, aplicando el resultado de (4) en (3) obtenemos que:

- Si n es par:

$$\begin{aligned} b^n \ \equiv \ x \ (n) \ , \quad con \ 0 \ \leq \ x \ < \ n \\ b^{n/2}.b^{n/2} \ \equiv \ x \ (n) \\ k.k \ \equiv \ x \ (n) \\ k^2 \ \equiv \ x \ (n) \end{aligned} \tag{5}$$

[1] Poner alguna referencia en donde se explique esta técnica

- Si n es impar par:

$$\begin{aligned} b^n \equiv x\ (n)\ , \quad con\ 0 \leq x < n \\ b.b^{n/2}.b^{n/2} &\equiv x\ (n) \\ b.k.k &\equiv x\ (n) \\ b.k^2 &\equiv x\ (n) \end{aligned} \tag{6}$$

Sin lugar a dudas, las expresiones resultantes en ambos casos son fáciles de resolver ya que $k < n$; al tiempo que también son números mucho menores que b^n.

Volviendo un poco sobre nuestro pasos, se dijo anteriormente en (4) que podíamos asumir que obtener k no tenía un costo computacional significativo. No obstante, esto no siempre es cierto puesto que para obtener k es necesario calcular previamente $b^{n/2}$ el cual puede ser un número nada despreciable en lo que respecta a tamaño en memoria.
Pero esto no representa, dificultad alguna para calcular $b^{n/2}$ puesto que podemos aplicar repetidas veces el procedimiento descripto con anterioridad dividiendo el exponente de b hasta que valga 1, y luego reagrupando los resultados y tomando módulo n hasta obtener finalmente la solución del problema.

2. Ejercicio 2

2.1. Introducción

El segundo problema consistió en la implementación de un algoritmo capaz de dar solución al problema que se plantea a continuación:

Decidir si un grupo de n personas pueden formar o no una ronda que cumpla con las siguientes restricciones:

- La ronda debe contener a todas las personas.
- Algunas personas son amigas y otras no.
- Cada alumna debe tomar de la mano a dos de sus amigas.

2.2. Explicación

Dado que para este problema no se conocen algoritmos buenos[2], se pensó en utilizar la solución por fuerza bruta, con algunas mejoras, es decir, intentar todas las combinaciones hasta lograr determinar si hay o no solución.

A este método, se le agregó

[2]Un algoritmo se considera bueno si puede ser resuelto en tiempo polinomial.

3. Ejercicio 3

3.1. Introducción

El tercer y último problema de este trabajo práctico consiste en, dadas dos listas con horarios de entrada y salida de ciertos trabajadores a una empresa, decidir cuál es el mayor número de los mismos que se encuentran al mismo tiempo dentro de dicha empresa.

En una primer mirada, se pensó que la mejor forma de resolver el ejercicio era aplicando un algoritmo similar al del quicksort[3], sólo que el mismo se realizaba sobre conjuntos y no sobre arreglos. En resumidas cuentas, la idea era tomar un trabajador cualquiera del grupo, y separar en tres conjuntos distintos: los que se cruzaban con el trabajador pivote, los que salían antes que el pivote entrara y los que entraban luego que el pivote saliera[4]. De esta manera, si se repetía el proceso varias veces, se llegaba a tener varios conjuntos en los cuales aparecían solamente trabajadores que se cruzaban en sus horarios dentro de la empresa. Finalmente, sólo restaba devolver el mayor cardinal de dichos conjuntos.

Al igual que el quicksort, si el pivote era elegido al azar, el algoritmo anterior contaba con una complejidad promedio de $O(n * log(n))$ (donde n es la cantidad de trabajadores). Igualmente, el peor caso seguía siendo $O(n^2)$, por lo que no se iba a poder respetar la cota dada como máxima para este trabajo, ya que se especificaba que el algoritmo debía tener una complejidad menor a $O(n^2)$.

Pero luego de un mejor análisis del problema, se cayó en la cuenta que los datos de entrada contaban con la característica de estar ordenados por horario (tanto los de entrada como los de salida), por lo que inmediatamente surgió la idea de poder solucionar el problema con una complejidad de $O(n)$.

Para lograr dicha solución, no se requirió de una técnica compleja. En realidad, lo que se hizo fue iterar sobre las dos listas de entrada, que contenían los horarios de entrada y de salida de los trabajadores. De esta forma, con un recorrido lineal sobre ambas listas de entrada, se podía saber con certeza cuál era la respuesta al problema para las mismas.

[3] Alguna referencia que explique el algoritmo

[4] Para que el algoritmo funcionara bien, se debía tener cierto cuidado con los trabajadores que se cruzaran con el pivote, pero la explicación de estos detalles no hacen a la escencia de la introducción. Además, este algoritmo fue descartado más adelante, por lo que estos detalles son irrelevantes para este trabajo.